Accelerated Kernel Feature Analysis

Xianhua Jiang, Student Member, IEEE, Robert R. Snapp, Member, IEEE, Yuichi Motai, Member, IEEE, and Xingquan Zhu, Member, IEEE

Abstract

A fast algorithm, Accelerated Kernel Feature Analysis (AKFA), that discovers salient features evidenced in a sample of \( n \) unclassified patterns, is presented. Like earlier kernel-based feature selection algorithms, AKFA implicitly embeds each pattern into a Hilbert space, \( H \), induced by a Mercer kernel. An \( \ell \)-dimensional linear subspace of \( H \) is iteratively constructed by maximizing a variance condition for the nonlinearly transformed sample. This linear subspace can then be used to define more efficient data representations and pattern classifiers. AKFA requires \( O(\ell n^2) \) operations, as compared to \( O(n^3) \) for Schölkof, Smola, and Müller’s Kernel Principal Component Analysis (KPCA), and \( O(\ell^2 n^2) \) for Smola, Mangasarian, and Schölkopf’s Sparse Kernel Feature Analysis (SKFA). Numerical experiments show that AKFA can generate more concise feature representations than both KPCA and SKFA, and demonstrate that AKFA obtains similar classification performance as KPCA for a face recognition problem.

I. INTRODUCTION

Automatic feature extraction has proven to be one of the most elusive problems in the field of pattern recognition. Although finding concise and informative representations of pattern data is an essential first step in designing a pattern recognition system, most research activity has centered on the more tractable tasks of developing and analyzing pattern classification algorithms[3]. Despite modest successes of principal component analysis [10], [7], [19], [11], self-organizing feature maps [8], and vector quantization [6] in the context of dimensionality reduction, feature selection is usually a manual process that demands an understanding of the problem domain (e.g., handwritten character or face recognition), as well as much trial and error.

This work was supported by the Vermont EPSCoR.
X. Jiang is with the University of Vermont.
Capitalizing on the success of kernel methods in pattern classification, Schölkopf, Smola and Müller [15] developed and studied a feature selection algorithm in which principal component analysis is effectively applied to a sample of \( n \), \( d \)-dimensional patterns that are first injected into a high-dimensional Hilbert space using a nonlinear embedding. (Heuristically, injecting input patterns into a high-dimensional space may elucidate salient nonlinear features in the input distribution, in the same way that nonlinearly separable classification problems may become linearly separable in higher dimensional spaces.) Both the principal component analysis and the nonlinear embedding are facilitated by a Mercer kernel of two arguments \( k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \), which effectively computes the inner product of the transformed arguments. This algorithm, called Kernel Principal Component Analysis (KPCA), thus avoids the problem of representing transformed vectors in the Hilbert space, and enables the computation of the inner-product of two transformed vectors of arbitrarily high dimension in constant time. Nevertheless, KPCA has two deficiencies: (i) the computation of the principal components involves the solution of an eigenvalue problem that requires \( O(n^3) \) computations, and (ii) each principal component in the Hilbert space depends on every one of the \( n \) input patterns, which defeats the goal of obtaining both an informative and concise representation.

Both of these deficiencies have been addressed in subsequent investigations that seek sets of salient features that only depend upon sparse subsets of transformed input patterns. Tipping [18] applied a maximum-likelihood technique to approximate the transformed covariance matrix in terms of such a sparse subset. Franc and Hlaváč [4] proposed a greedy method, which approximates the mapped space representation by selecting a subset of input data. It iteratively extracts the data in the mapped space until the reconstruction error in the mapped high-dimensional space falls below a threshold value. Its computational complexity is \( O(nm^3) \), where \( n \) is the number of input patterns, and \( m \) is the cardinality of the subset. Zheng [21] split the input data into \( M \) groups of similar size, and then applied KPCA to each group. A set of eigenvectors was obtained for each group. KPCA was then applied to a subset of these eigenvectors to obtain a final set of features. Although these studies propose useful approaches, none provides a method that is both computationally efficient and accurate.

To avoid the \( O(n^3) \) eigenvalue problem, Smola, Mangasarian and Schölkopf [17] proposed Sparse Kernel Feature Analysis (SKFA), which extracts \( \ell \) features, one by one, using an \( l_1 \) constraint on the expansion coefficients. SKFA requires only \( O(\ell^2n^2) \) operations, and is thus a
significant improvement over KPCA if number of dominant features is much less than the data size. However, if \( \ell > \sqrt{n} \), then the computational cost of SKFA likely exceeds that of KPCA. Schölkopf and Smola [16] introduced two efficient approximations of SKFA: (i) a “Probabilistic Speedup”, which approximates the sums by computing only a subset of the terms, and (ii) a Quantile Trick, which computes a random subsample of all possible directions. Both methods reduce the accuracy of SKFA.

In the following, we propose an Accelerated Kernel Feature Analysis (or AKFA), which generates \( \ell \) sparse features from a data set of \( n \) patterns using \( \mathcal{O}(\ell n^2) \) operations. Since AKFA is based on both KPCA and SKFA, we analyze the former algorithm in Section 2, and the latter in Section 3. AKFA is then described and analyzed in Section 4. (We also describe a compressed approximation of AKFA, called ACKFA, which terminates early.) Section 5 summarizes our numerical experiments which (i) confirm the run-time complexity of each of the four algorithms, (ii) demonstrate that AKFA can generate more informative and concise features than KPCA and SKFA, and (iii) illustrate that AKFA obtains similar classification performance as KPCA for a face recognition problem. Our conclusions appear in Section 6.

II. KERNEL PCA (KPCA)

Using Mercer’s theorem (cf., [2]), a nonlinear, positive-definite kernel \( k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \) of an integral operator, e.g., \( k(x, y) = \exp\{-\|x - y\|^2\} \), computes the inner product of the transformed vectors \( \langle \Phi(x), \Phi(y) \rangle \), where \( \Phi : \mathbb{R}^d \rightarrow \mathcal{H} \) denotes a nonlinear embedding (induced by \( k \)) into a possibly infinite dimensional Hilbert space, \( \mathcal{H} \). Given \( n \) points \( \mathcal{X}_n = \{x_i \in \mathbb{R}^d | i = 1, \ldots, n\} \), the image \( \mathcal{Y}_n = \{\Phi(x_i) | i = 1, \ldots, n\} \) of \( \mathcal{X}_n \) spans a linear subspace of at most \( (n - 1) \) dimensions. Heuristically, the dominant linear correlations in the distribution of the image may elucidate important nonlinear dependencies in the original data sample \( \mathcal{X}_n \).

Kernel Principal Component Analysis (KPCA) uses a Mercer kernel to perform a linear principal component analysis of this transformed image. Without loss of generality, we assume that the image of the data has been centered so that its scatter matrix in \( \mathcal{H} \) is given by \( S = \sum_{i=1}^{n} \Phi(x_i)\Phi(x_i)^\top \). Eigenvalues \( \lambda_j \) and eigenvectors \( e_j \) are obtained by solving

\[
\lambda_j e_j = Se_j = \sum_{i=1}^{n} \Phi(x_i)\Phi(x_i)^\top e_j = \sum_{i=1}^{n} \langle e_j, \Phi(x_i) \rangle \Phi(x_i),
\] (1)
for \( j = 1, \ldots, n \). Since \( \Phi \) is not known, (1) must be solved indirectly. Letting, \( a_{ji} = \frac{1}{\lambda_j} \langle e_j, \Phi(x_i) \rangle \), (1) becomes
\[
e_j = \sum_{i=1}^{n} a_{ji} \Phi(x_i). \tag{2}\]

Multiplying by \( \Phi(x_q)^T \) on the left, for \( q = 1, \ldots, n \), yields
\[
\lambda_j \langle \Phi(x_q), e_j \rangle = \sum_{i=1}^{n} \langle e_j, \Phi(x_i) \rangle \langle \Phi(x_q), \Phi(x_i) \rangle. \tag{3}\]

Substitution of (2) into (3) produces
\[
\lambda_j \langle \Phi(x_q), \sum_{i=1}^{n} a_{ji} \Phi(x_i) \rangle = \sum_{i=1}^{n} \left( \langle \sum_{k=1}^{n} a_{jk} \Phi(x_k), \Phi(x_i) \rangle \langle \Phi(x_q), \Phi(x_i) \rangle \right), \tag{4}\]
which can be rewritten as, \( \lambda_j K a_j = K^2 a_j \), where \( K \) is a \( n \times n \) Gram matrix, with the element \( k_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle \), and \( a_j = [a_{j1} \ a_{j2} \ \cdots \ a_{jn}]^T \). The latter is a dual eigenvalue problem equivalent to the problem
\[
\lambda_j a_j = K a_j. \tag{5}\]

Since \( \|e_j\|^2 = \left\langle \sum_{i=1}^{n} a_{ji} \Phi(x_i), \sum_{i=1}^{n} a_{ji} \Phi(x_i) \right\rangle = \langle a_j, K a_j \rangle = \lambda_j \|a_j\|^2 \), the normalization of each eigenvector (\( \|e_j\| = 1 \)) requires that \( \|a_j\|^2 = 1/\lambda_j \).

In the following, we choose a Gaussian kernel, i.e.,
\[
k_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = \exp \left( -\frac{1}{2\sigma^2} \|x_i - x_j\|^2 \right). \tag{6}\]

If the image of \( \mathcal{X}_n \) is not centered in the Hilbert space, we need to use the centered Gram Matrix deduced by Smola, Mangasarian and Schölkoph [17]:
\[
\hat{K} = K - KT - TK + TKT \tag{7}\]

where \( K \) is the Gram Matrix of uncentered data, and
\[
T = \begin{bmatrix}
\frac{1}{n} & \ldots & \frac{1}{n} \\
\ldots & \ldots & \ldots \\
\frac{1}{n} & \ldots & \frac{1}{n}
\end{bmatrix}_{n \times n}
\]

Keeping the \( \ell \) eigenvectors associated with the \( \ell \) largest eigenvalues, we can reconstruct data in the mapped space:
\[
\Phi_i' = \sum_{j=1}^{\ell} \langle \Phi_i, e_j \rangle e_j = \sum_{j=1}^{\ell} \beta_j e_j,
\]
where $\beta_{ji} = \langle \Phi_i, \sum_{k=1}^{n} a_{jk} \Phi_k \rangle = \sum_{k=1}^{n} a_{jk} k_{ik}$. The reconstruction square error of each data $\Phi_i, i = 1, \ldots, n$, is

$$\text{Err}_i = \| \Phi_i - \Phi'_i \|^2 = k_{ii} - \sum_{j=1}^{\ell} \beta_{ji}^2.$$  

The mean square error is $\text{MErr} = \frac{1}{n} \sum_{i=1}^{n} \text{Err}_i$. Using (5), $\beta_{ji} = \lambda_j a_{ji}$. Therefore, the mean square reconstruction error is $\text{MErr} = \frac{1}{n} \sum_{i=1}^{n} (k_{ii} - \sum_{j=1}^{\ell} \lambda_j^2 a_{ji}^2)$. Since $\sum_{i=1}^{n} k_{ii} = \sum_{i=1}^{n} \lambda_i$, and $\sum_{i=1}^{n} a_{ji}^2 = \| a_j \|^2 = 1/\lambda_j$, $\text{MErr} = \frac{1}{n} \sum_{i=\ell+1}^{n} \lambda_i$.

Kernel PCA can now be summarized: First, calculate the Gram matrix using (6), which contains the inner products between pairs of image vectors. Then use (5) to get the coefficient vectors $a_j$, for $j = 1, \ldots, n$. The projection of a test point $x \in \mathbb{R}^d$ along the $j$-th eigenvector is

$$\langle e_j, \Phi(x) \rangle = \sum_{i=1}^{n} a_{ji} \langle \Phi(x_i), \Phi(x) \rangle = \sum_{i=1}^{n} a_{ji} k(x, x_i).$$

Because the above implicitly contains an eigenvalue problem of rank $n$, the computational complexity of Kernel PCA is $O(n^3)$. In addition, each resulting eigenvector is represented as a linear combination of $n$ terms. Thus, all data contained in $X_n$ must be retained, which is computationally cumbersome, and unacceptable for incremental learning.

III. SPARSE KERNEL FEATURE ANALYSIS (SKFA)

As mentioned in the introduction, several investigations have addressed the computational costs of KPCA, associated with both time complexity and data retention requirements. In particular, Sparse Kernel Feature Analysis (SKFA) [17] extracts the features one by one in order of decreasing projection variance. Another advantage is that the $\ell$ features of SKFA only depend on $\ell$ elements of $X_n$, which is extremely useful for on-line learning.

In the following, we let $v_i \in H$, for $i = 1, \ldots, \ell$ denote the features selected by SKFA. (We intentionally avoid using $e_i$, in order to distinguish these features from the eigenvectors obtained using KPCA.) Again, we analyze the scatter matrix of the image data. Following (1), with $e_j$ replaced by $v_j$, we obtain $v_j^\top \lambda_j v_j = \langle v_j, (\sum_{i=1}^{n} \langle v_j, \Phi(x_i) \rangle) \Phi(x_i) \rangle$, where $v_j$ is the $j$th feature with unit length. Thus, $\lambda_j = \sum_{i=1}^{n} \langle v_j, \Phi(x_i) \rangle^2$. Therefore, the first feature, corresponding to the maximum eigenvalue, is chosen as the direction with the maximum projected variance:

$$v_1 = \arg \max_{\| v \|^2 = 1} \frac{1}{n} \sum_{i=1}^{n} | \langle v, \Phi(x_i) \rangle |^2.$$  

April 20, 2006
The global solution of (8), $v_1$, needs to satisfy an $l_2$ normalization constraint, i.e., unit Euclidian length. Changing the $l_2$ constraint to an $l_1$ constraint leads to a vertex solution. The $l_1$ constraint assumed by SKFA is

$$V_{l_1} = \left\{ \sum_{j=1}^{n} a_j \Phi_j \left| \sum_{j=1}^{n} |a_j| \leq 1 \right. \right\}. \quad (9)$$

The first feature selected by SKFA satisfies

$$v_1 = \operatorname{arg\,max}_{v \in V_{l_1}} \frac{1}{n} \sum_{i=1}^{n} |\langle v, \Phi(x_i) \rangle|^2.$$

Smola et al. showed that this feature corresponds to an element of the image $Y_n$, whence

$$v_1 = \operatorname{arg\,max}_{\Phi(x_j) \in V_{l_1}} \frac{1}{n} \sum_{i=1}^{n} |\langle \Phi(x_j), \Phi(x_i) \rangle|^2. \quad (10)$$

Subsequent features are obtained iteratively. Suppose, $i-1$ features $\{v_t \in H | t = 1, \ldots, i-1\}$ have been found, then project each image $\Phi_j = \Phi(x_j)$ into the orthogonal subspace to obtain

$$\Phi^i_j = \Phi_j - \sum_{t=1}^{i-1} v_t \frac{\langle \Phi_j, v_t \rangle}{\|v_t\|^2} = \Phi_j - \sum_{t=1}^{i-1} \frac{a_{tj} v_t}{\|v_t\|^2}, \quad (11)$$

where $a_{tj} = \langle \Phi_j, v_t \rangle$. Then normalize $\Phi^i_j$ by the $l_1$ constraint in (9). The projection variance of the normalized $\Phi^i_j$ with all $\Phi_k, k = 1, \ldots, n$ is then calculated. Finally one identifies the maximum projection variance and selects the corresponding $\Phi^i_j$ as the $i$th feature, $v_i$.

Based on the $\ell$ features extracted by SKFA, each training data in the mapped space can be reconstructed by

$$\Phi'_i = \sum_{j=1}^{\ell} \frac{\langle \Phi_i, v_j \rangle v_j}{\|v_j\|^2} = \sum_{j=1}^{\ell} \frac{a_{ji} v_j}{\|v_j\|^2}, \quad (12)$$

where $a_{ji}$, the projection of $i$-th training data on $j$-th feature, is stored after extracting the $j$-th feature. According to (11), the feature set $\{v_1, \ldots, v_\ell\}$ only depends upon the set of $\ell$ image vectors, $\{\Phi_{\text{idx}(i)}, i = 1, \ldots, \ell\}$, where $\text{idx}(i)$ denotes the subscript of the projected image $\Phi^i_j$ that was selected when constructing $v_i$. Therefore, after training, we only need to retain the $\ell$ input vectors $\{x_{\text{idx}(i)} \in \mathbb{R}^d | i = 1, \ldots, \ell\}$, where $\ell$ is the number of features extracted.

In this way, SKFA extracts $\ell$ features, where one assumes $\ell \ll n$. As $O(in^2)$ operations are required to extract the $i$-th feature, the total computational cost for $\ell$ features is $O(\ell^2 n^2)$, which is an improvement over the $O(n^3)$ operations required by kernel PCA.
IV. ACCELERATED KERNEL FEATURE ANALYSIS (AKFA)

To improve the efficiency and accuracy of SKFA, we propose an Accelerated Kernel Feature Analysis (AKFA) that (i) saves computation time by iteratively updating the Gram Matrix, (ii) normalizes the images with the $l_2$ constraint before the $l_1$ constraint is applied, and (iii) optionally discards data that falls below a threshold magnitude $\delta$ during updates.

First, instead of extracting features directly from the original mapped space, AKFA extracts the $i$-th feature based on the $i$-th updated Gram matrix $K^i$, where each element $k^i_{jk} = \langle \Phi^i_j, \Phi^i_k \rangle$.

By updating the Gram Matrix, we don’t need to save the projection of each individual data on all previous features. The computational cost for extracting $i$-th feature becomes $O(n^2)$, instead of $O(in^2)$ as in SKFA.

The second improvement is to revise the $l_1$ constraint. As shown in (10), SKFA treats each individual sample data as a possible direction, and computes the projection variances with all data. Since SKFA includes its length in its projection variance calculation, it is biased to select vectors with larger magnitude. From the objective function (8), we know that we are actually looking for a direction with unit length. When we choose a image vector as a possible direction, we only consider its direction ignoring the length, which improves the accuracy of the features.

Therefore, in our AKFA algorithm, we replace the $l_1$ constraint (9) by

$$V^i_{l_1} = \left\{ \sum_{j=1}^{n} a_j \frac{\Phi^i_j}{\|\Phi^i_j\|} \left| \sum_{j=1}^{n} |a_j| \right| \leq 1 \right\}.$$

The $i$-th feature is extracted by

$$v_i = \arg \max_{v \in V^i_{l_1}} \frac{1}{n} \sum_{j=1}^{n} |\langle v, \Phi^i_j \rangle|^2.$$
Since \( v_i \) is extracted from \( \Phi^i \) space, the solution is located on one of the \( \hat{\Phi}^j = \Phi^j / \| \Phi^j \| \) for \( j = 1, \ldots, n \). Equation (10) reduces to
\[
v_i = \arg \max_{\Phi_j} \frac{1}{n} \sum_{t=1}^{n} \| (\Phi^i_t, \hat{\Phi}^j) \|^2 = \arg \max_{\Phi_j} \frac{1}{n} \sum_{t=1}^{n} k_{jt}^2.
\] (14)

Each \( \hat{\Phi}^j \) satisfies the \( l_1 \) constraint, so the normalization step that appears in SKFA is not required.

Let \( \Phi^i_{\text{idx}(i)} \) denote the image vector corresponding to the \( i \)-th feature. Suppose we have selected \((i - 1)\) features with \( V_{(i-1)} = \Phi_{(i-1)} C_{(i-1)} \), where \( V_{(i-1)} = [v_1 \ v_2 \ \cdots \ v_{(i-1)}] \), \( \Phi_{(i-1)} = [\Phi_{\text{idx}(1)} \ \Phi_{\text{idx}(2)} \ \cdots \ \Phi_{\text{idx}(i-1)}] \), and \( C_{(i-1)} \) is the coefficient matrix, which is upper-triangular. Then \( \Phi^i_{\text{idx}(i)} = \Phi^i_{\text{idx}(i)} - \sum_{t=1}^{i-1} (\Phi^i_{\text{idx}(i)}, v_t) v_t \). Let’s study the second term:
\[
\sum_{t=1}^{i-1} (\Phi^i_{\text{idx}(i)}, v_t) v_t = \sum_{t=1}^{i-1} v_t v_t^T \Phi^i_{\text{idx}(i)} = \Phi^i_{(i-1)} C_{i-1} C_{i-1}^T \Phi^i_{\text{idx}(i)}
\] (15)

where \( K_{\text{idx}(i)} = [k_{\text{idx}(i),\text{idx}(1)} k_{\text{idx}(i),\text{idx}(2)} \cdots k_{\text{idx}(i),\text{idx}(i-1)}]^T \). Therefore,
\[
v_i = (\Phi^i_{\text{idx}(i)} - \Phi^i_{(i-1)} C_{i-1} C_{i-1}^T K_{\text{idx}(i)}) / \sqrt{k_{i,\text{idx}(i),\text{idx}(i)}}
\]

Let
\[
C_{i,i} = 1 / \sqrt{k_{i,\text{idx}(i),\text{idx}(i)}},
\] (16)
and
\[
C_{1:(i-1),i} = -C_{i,i} C_{(i-1)} C_{i-1}^T K_{\text{idx}(i)};
\] (17)

then \( V_i = \Phi_i C_i \), where \( V_i = [V_{(i-1)}, v_i] \), and \( \Phi_i = [\Phi_{(i-1)}, \Phi^i_{\text{idx}(i)}] \), \( C_i = \begin{pmatrix} C_{(i-1)} & C_{1:(i-1),i} \\ 0 & C_{i,i} \end{pmatrix} \).

The third improvement is to discard negligible data and thereby eliminate unnecessary computations. In the \( i \)-th updated Gram matrix \( K^i \), defined in (13), the diagonal elements \( k_{jj}^i \) represents the reconstruction error of the \( j \)-th data point with respect to the previous \((i - 1)\) features. If \( k_{jj}^i \) is relatively small, then the previous \((i - 1)\) features contain most of the information that would be acquired from this image vector. One can therefore optionally discard image points that satisfy \( k_{jj}^i < \delta \), for some predetermined \( \delta > 0 \). In the following, we call this variation Accelerated Cut-off Kernel Feature Analysis (ACKFA). ACKFA can be useful if the data size is huge. It can also be used as a criteria to stop extracting features if one is unsure of the number of features that should be selected.
The entire algorithm (ACKFA) is summarized below:

Step 1 Compute the \( n \times n \) Gram matrix \( k_{ij} = k(x_i, x_j) \), where \( n \) is the number of input vectors. This part requires \( O(n^2) \) operations.

Step 2 Let \( \ell \) denote the number of features to be extracted. Initialize the \( \ell \times \ell \) coefficient matrix \( C \) to 0, and \( \text{idx}(\cdot) \) as an empty list which will ultimately store the indices of the selected image vectors. Initialize the threshold value \( \delta \) for the reconstruction error. (For AKFA, set \( \delta = 0 \).) The overall cost is \( O(\ell^2) \).

Step 3 For \( i = 1 \) to \( \ell \) repeat:

1) Using the \( i \)-th updated \( K^i \) matrix, extract the \( i \)-th feature using (14). If \( K^i_{jj} < \delta \), then discard \( j \)th column and \( j \)th row vector without calculating the projection variance. Use \( \text{idx}(i) \) to store the index. This step requires \( O(n^2) \) operations.

2) Update the coefficient matrix by using (16) and (17), which requires \( O(i^2) \) operations.

3) Use (13) to obtain \( K^{i+1} \), an updated Gram matrix. Neglect all rows and columns that contain a diagonal element less than \( \delta \). This step requires \( O(n^2) \) operations.

The total computational complexity of ACKFA is up to \( O(\ell n^2) \) when no data is cut during updating, which is the same as AKFA. If we increase \( \delta \), more data will be cut, and the total computational time will be decreased. Since the number of data being cut at each step depends on \( \delta \) and data set, we consider an average situation. Suppose after extracting a feature, we only keep a fraction \( p \) of all possible directions, and let there be one point left after extracting \( \ell \) features. That means \( np^\ell = 1 \), where \( n \) is the data size. It is equal to \( p = n^{-1/\ell} \). The total computational time is

\[
CT_{ACKFA} = \sum_{k=0}^{\ell-1} (np^k)^2 = \frac{n^2 - 1}{1 - n^{-2/\ell}} \approx \frac{n^2}{1 - n^{-2/\ell}} \tag{18}
\]

when \( n \) is large. The computational time of SKFA is \( CT_{SKFA} = O(\ell^2 n^2) \). Therefore,

\[
\frac{CT_{SKFA}}{CT_{ACKFA}} = \ell^2 (1 - n^{-2/\ell}) \tag{19}
\]

For example, when \( \ell = 50 \), data size \( n = 3000 \), \( CT_{SKFA}/CT_{ACKFA} \) is about 684, so the speed of ACKFA is 684 times faster than SKFA! The experimental results will confirm that our features have better performance than those obtained by SKFA.
V. EXPERIMENTS

We check the performance of the features based on the computational time, reconstruction error in the mapped space, projection variance, and face recognition. We use mean square error to represent the reconstruction error, and a Gaussian kernel with \( \theta = 4 \), for Experiments 5.1 to 5.3. All results were obtained using Matlab 7.0.1 on a Dell Inspiron 600M with Intel Pentium(R) M processor 1.5GHz and 512 MB of RAM. We use the Statistical Pattern Recognition Toolbox [5] for the Gram matrix calculation and KPCA algorithm.

A. Computational Time and Reconstruction Error with Data Size

In this experiment, we use an artificially generated data set in \( \mathbb{R}^2 \). The data form a circle centered at \((0,0)\) with radius 8.0. The data are corrupted by adding normally distributed noise with standard variance 1.0. The data size is increased from 500 to 3000. For each algorithm AKFA, ACKFA with \( \delta = 0.4 \), KPCA, and SKFA, we extract ten features that describe each data set. The results are shown in Table I, II, and Figure 1. Table I indicates that the computational time of KPCA increases rapidly with the data size \( n \). When \( n = 3500 \), the computational time of KPCA is a factor of 293 larger than that of ACKFA, about 147 times that of AKFA, and about 2 times of SKFA. The computational time versus data size \( n \) is plotted in Figure 1 with common-logarithm scales. The following linear fits validate the complexity analyses in Sections 2 through 4.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>( \log_{10}(\text{time}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>KPCA</td>
<td>( 3.03 \log_{10} n - 7.45 )</td>
</tr>
<tr>
<td>SKFA</td>
<td>( 2.05 \log_{10} n - 4.32 )</td>
</tr>
<tr>
<td>AKFA</td>
<td>( 2.00 \log_{10} n - 6.02 )</td>
</tr>
<tr>
<td>ACKFA</td>
<td>( 2.01 \log_{10} n - 6.36 )</td>
</tr>
</tbody>
</table>

Note that SKFA consumes more time than KPCA when \( n \) is small and \( \ell \), the number of features is large. For example, when data size is 300, the number of features is 50, the computational time of SKFA with \( O(\ell^2 n^2) \) is about 8 times that of KPCA with \( O(n^3) \).

Table II shows that KPCA, which computes accurate estimates of the eigenvectors, incurs a smaller reconstruction error than the other approximate methods. In this sense, the features generated by KPCA are more accurate. The reconstruction errors appear to be stable for all methods as \( n \) increases.
B. ACKFA with $\delta$

The previous experiment reveals a trade off between reconstruction error and computation time for ACKFA and AKFA. Using the same problem, with $n = 1000$, we now study the performance of ACKFA with respect to $\delta$. With $\ell = 20$ features, we increase $\delta$ from 0.01 to 0.15. If $\delta$ is larger than 0.15, all data are cut off before we reach 20 features. Without cutting, the computational time is 1.8326 s, and the reconstruction error is 0.025. The computational time and reconstruction error of ACKFA are plotted against $\delta$ in Figure 2. The computational time decreases almost linearly with respect to $\delta$. In contrast, the reconstruction error remains steady for $0.05 \leq \delta \leq 0.07$, then increases rapidly from $0.07 \leq \delta \leq 0.11$, and flattens out from $0.11 \leq \delta \leq 0.15$. A careful choice of $\delta$ can balance the trade off between efficiency and accuracy.

C. Interpreting Features using Projection Variance

This experiment uses the objective function (8) to verify the features by using KPCA, ACKFA with $\delta = 0.4$, AKFA and SKFA. Using the artificial data set as before, random data sets from size $n = 500$ to $n = 3000$ are generated. For each method, we calculated the projection variance...
on the first 10 features $\frac{1}{n} \sum_{i=1}^{10} \sum_{j=1}^{n} \langle \Phi(x_j), v_i \rangle^2$, which measures the total information those 10 features contain. The experimental results in Figure 3 show that KPCA captures the maximum projection. AKFA and ACKFA outperform SKFA, which we attribute to the modification of (10). AKFA is better than ACKFA because the latter neglects data points.

D. AKFA and KPCA

Because AKFA is significantly faster than KPCA, one may be able to extract more features to reach or exceed the accuracy of KPCA in the same amount of time. To test this hypothesis,
we generate the same type of random pseudo data set as before with size 1000 in $\mathbb{R}^2$. KPCA requires 38.3552 s to get 10 features, with a reconstruction error of 0.0554. If we use AKFA without cutting data, the accuracy and computational time with the number of features is shown in Table III. When number of features is 20, the time is 1.84s, the error is 0.0144, which is around quarter of KPCA. When number of features is 60, AKFA spends about 1/7 time of KPCA, and the accuracy is about 1000 times of KPCA. The size of coefficient matrix is $60 \times 60$, while the coefficient matrix of KPCA is $10 \times 1000$. Therefore, to reach the accuracy of KPCA, AKFA can extract more features, while still use less time and storage.
E. Pre-image

A typical application area of KPCA is de-noising input data [12], [13], [9], [14]. The basic procedure is to (i) obtain the dominant features in the image space, then (ii) project a noisy test vector onto this reduced subspace (i.e., reconstruct the test vector in the subspace), and then (iii) find a point (called the pre-image) in the input space that has an image close to that of the reconstructed test vector. Generally, it’s difficult to find an exact pre-image, Mika et al. [13] used fixed-point iteration to approximate the pre-image for a Gaussian kernel, while Kwok and Tsang [9] used a distance constraint in the image space. We have adopted the latter method, as it appears to be more efficient and stable.

1) Pre-image of Pseudo Data Set: We artificially generate 1200 points in the shape of a Swiss Cross, centered at \((0,0)\) in \(\mathbb{R}^2\). Normally distributed noise with variance 1.0 is added to each vector. First, we extract 15 features in the image space, then project each image vector to the feature subspace. Then we find a pre-image for each reconstructed image vector, and compare the error between the pre-image and the true data without noise. The mean error between the pre-image and the true data, and the computational time, are compared between KPCA, AKFA and SKFA:
Obviously, AKFA spends much less time than KPCA and SKFA. From the pre-images shown in Figure 4, KPCA has the best de-noise performance. AKFA is very competitive with KPCA and much better than SKFA. From the data points extracted by AKFA and SKFA, we can see that basically both methods are picking the corner points. But SKFA prefers “extreme points”, while AKFA looks for points close to true data. This difference stems from the different way each method calculates the projection variance. This experiment suggests that AKFA extracts more informative features than SKFA.

2) Pre-image of USPS Data Set: Another pre-image experiment uses USPS zip code. Each image has $16 \times 16$ gray-level pixels. There are totally 10 classes corresponding to 10 digits. To each digit, we select 300 sample images as a training data set. Each sample image is corrupted by adding normally distributed noise with standard variance 0.5 to each pixel. We extract 5 features from each class. Then, we find the pre-image for each reconstructed sample. The result is shown in Figure 5. We can see that all methods can de-noise very well, and KPCA and SKFA are slightly better than AKFA.

F. USPS Data set Classification

The most important motivation for extracting features is to classify the data. From the probability of linear separability for dichotomies $P(m, n) = \frac{1}{2^{m-1}} \sum_{i=0}^{n} C_{m-1}^{i}$, where $m$ is the number of patterns, $n$ is the dimension of patterns, we can see that when $m$ is fixed, higher dimensional space has better classification performance than lower dimensional space. Therefore, for those nonlinear separable data in the low dimensional input space, we mapped them to the high dimensional Hilbert space to get the better classification performance. However, in the high dimension space, we need to face the Bellman’s curse of dimensionality, that means higher dimensional space needs much more samples from the aspect of generalization. Another disadvantage is the increased computational complexity. Thus, we use AKFA to extract a reduced feature space, then classify the data in this small feature space.

The classification method we used here is $k$-nearest neighbor, which is simple yet has nearly optimal performance in the large sample limit. For each test point, we find its $k$ nearest labeled
Fig. 4. Comparison between KPCA, AKFA and SKFA for the Swiss Cross problem. 1200 x-shaped markers depict the noise samples. Circular markers depict the de-noised data using KPCA in (b), AKFA in (c) and SKFA in (d). Diamonds in (c) and (d) denotes the data points that were chosen to construct the subspace.

sample, then use majority voting to decide the class of this test point.

The data set we use is USPS zip code. First, we use 3000 training data, 400 test data that are not included in the training samples, 8 nearest neighbors, the kernel we use is Gaussian kernel with $\sigma = 4$. The dimension of the reduced feature space is increased from 10 to 1000. The accuracy of the classification with respect to the number of features is shown in Figure 6. We can see that the accuracy is first increased, then reach stable, finally decreased. Then, we keep
the number of features fixed at 100, but change the number of nearest neighbors from 1 to 50. Figure 7 shows that the accuracy is decreased with the increase of the neighbor. The third sub experiment is to change the coefficient of kernel, while keep 100 features, 1-nearest neighbor. Figure 8 shows that the best classification performance is reached when $\sigma = 10$, then decreases again. Finally, we change the size of the training data set. Still set $\sigma = 4$, 100 features, 1-nearest neighbor. Figure 9 shows that there is a jump from size 2500 to 3000.

The distribution of the data in the extracted subspace is shown in Figure 10. We can see that digit 8 is mixed with the other three digits, while other three digits are well separated. We need to note that these coefficients are not totally independent, therefore combining the best parameter from each experiment doesn’t guarantee the best accuracy. For example, when $\sigma = 10$, 1-nearest neighbor, 300 features, the accuracy is 93.0%, which is worse than the case for 100 features. The best accuracy reported for this data set using Kernel SVM is 95.7%. However, they used all 7291 training samples, while we only used 3000 of them.
Fig. 6. Accuracy of k-nearest neighbor classifier with respect to the dimension of the reduced feature space. USPS data set.
3000 training data, 400 test data, 8 nearest neighbors, rbf kernel with $\sigma = 4$.

G. Face Recognition

Previous research demonstrates that features obtained by kernel analysis, including KPCA, are useful for face recognition applications [20]. The following experiments demonstrate that AKFA is also useful within this problem domain. Following Yang et al., the Yale Face Database [1] is classified using a k-nearest neighbor classifier under the leave-one-out method. The data set consists of 165 gray-scale images of 15 different human subjects, each with 11 exposures: center-light, w/glasses, happy, left-light, w/o glasses, normal, right-light, sad, sleepy, surprised, and wink. Each image is down-sampled to a $26 \times 25$ pixel array, and is subsequently centered and normalized. In each trial, one of the images is left out, and the remaining 164 are analyzed by AKFA, ACKFA with $\delta = 0.16$, SKFA, and KPCA with a Gaussian kernel with width $\sigma \in \{14, 20, 24, 34\}$.

In the first experiment, we want to investigate the classification performance with the respect of the number of features. We set $k = 10, \sigma = 24$. Figure 11 compares the accuracies of the four algorithms for $\ell$ from 10 to 100. It shows that the accuracies first increase rapidly, and then
Fig. 7. Accuracy of k-nearest neighbor classifier with respect to the number of neighbors. USPS data set. 3000 training data, 400 test data, 100 features, rbf kernel with $\sigma = 4$.

...approach stable. In the later experiments, we choose $\ell = 80$ when all four algorithms have good performance.

In the second experiment we select $k = 10$, and compare in Table IV the resulting classification accuracy and execution time of these four algorithms using the four different values of $\sigma$. (Since $\ell^2 \gg n$, SKFA requires significantly more time than KPCA.) Figure 12 compares the accuracies of the four algorithms for a range of $k$, with $\sigma = 24$. The best accuracy occurs when the $k$ is close to 10.

With $k = 10$ and $\sigma = 24$, confusion matrix is constructed for AKFA in Tables V, which shows the error frequencies of each class (depicted in bold), according to class. The confusion matrices generated by KPCA and SKFA are similar. In the event of a tie, the most frequent class with the lowest index is selected. The seventh (right-light) exposure of Subject 1 is incorrectly selected an inordinate number of times. Faces from Subjects 8 and 12, are also difficult to classify. (These images are shown in Figure 13).

For comparison, the best accuracy obtained with features selected by linear PCA, is 60.61%,
Fig. 8. Accuracy of k-nearest neighbor classifier with respect to the coefficient of RBF kernel. USPS data set. 3000 training data, 400 test data, 1-nearest neighbors, 100 features.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>$\sigma$</th>
<th>TIME(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>AKFA(%)</td>
<td>72.12</td>
<td>75.15</td>
</tr>
<tr>
<td>ACKFA(%)</td>
<td>72.12</td>
<td>76.36</td>
</tr>
<tr>
<td>KPCA(%)</td>
<td>76.36</td>
<td>75.76</td>
</tr>
<tr>
<td>SKFA(%)</td>
<td>68.48</td>
<td>74.55</td>
</tr>
</tbody>
</table>

TABLE IV

**Classification accuracy with the Gaussian kernel parameter $\sigma$**

which corresponds to a subspace of $\ell = 50$ dimensions and $k = 5$. Consequently, kernel features can improve the performance of face recognition. Although the features of AKFA have a larger reconstruction error than KPCA, the two methods have similar performance for face recognition, which further confirms that the features of AKFA are useful for computer vision and pattern recognition.
VI. CONCLUSION

This paper describes, analyzes, and demonstrates AKFA, a new feature extraction algorithm derived from the Sparse Kernel Feature Analysis (SKFA) of Smola et al [17]. The time complexity of AKFA is $O(\ell n^2)$ which is more efficient than the $O(\ell^2 n^2)$ time complexity of SKFA, and the complexity $O(n^3)$ of a more systematic principal component analysis (KPCA). The face recognition experiment shows that AKFA has the similar classification performance as KPCA using $k$-nearest neighbor classifier, which demonstrates that the features extracted by AKFA are practically useful.

We hope this work will inspire future studies that reveal how AKFA can be used to elucidate hidden structures in practical vision problems, such as image retrieval. We intend to study how the accuracy of AKFA can be further improved by representing features as more general linear combinations of image vectors.

Fig. 9. Accuracy of k-nearest neighbor classifier with respect to the number of samples. USPS data set. rbf kernel with $\sigma = 4$, 400 test data, 1-nearest neighbors, 100 features.
Fig. 10. RBF kernel with $\sigma = 4$, 400 test data, 3000 training data. 1-nearest neighbors, 3 features.

REFERENCES


Fig. 11. Accuracy with the Number of Feature

Fig. 12. Accuracy with the Number of Nearest Neighbor
### Table V

**Confusion Matrix of AKFA**

<table>
<thead>
<tr>
<th>True Pattern Class</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fig. 13. Top Layer: Subject 1; Middle Layer: Subject 8; Bottom Layer: Subject 12.


